

28/8/23

①

Things I know about computers

- They're smart and have no iq
- They can perform complex and accurate calculations
- They can multitask
- They can be used to improve our day to day tasks
- They can make our lives easier
- Various everyday tasks can be achieved/finished using them
- They require electricity to work

What is a computer?

An electronic device that takes in some input from the user, processes it and gives output. It's different from electrical devices, less voltage required

- ~~There are 3 types~~ Programming language acts as a layer for communication
- A programming language is required to communicate w/ computer
- Without it, we can't communicate with computer
- Purpose of communicating with computer is to assign the work
- Work maybe adding of 2 numbers, multiplication of 2 nos, addition of 2 matrices; Finding roots of quadratic eq.
- We need to give a name to the work
 - 1) Addition - of - 2 - nos
 - 2) Addition - of - 2 - matrices

- Machine can understand a form of 0s & 1s
 - We can use zeros & ones in a language \rightarrow machine language or binary or low level language
- Ex:- $C = 1 + 2$, in this language C has to be represented using approx 8 bits
 $=$ is also 8 bits, $1 \rightarrow$ 8 bits, $2 \rightarrow$ 8 bits

- Medium level language:- This language is also called assembly language. In this language, we use some symbols like add, sub, mul... to minimize the complexity of machine language

- High level language:- We use English usage letters and words
- Ex:- C, C++, Java

- Program:- A set of instructions to perform a particular task

- Instructions:- It is a valid statement accepted by the computer interpreter and produces the result

- Software:- Software is collection of programs

Two types of software:-

1) System software

2) Application software

- System software: Software which provides an environment in which apps can be executed
- without system software, there is no use of application software
ex: operating system
- Application software: Is used to solve user's problems
ex: MS office, AutoCAD

Basic structure of a C program

Documentation section

link section (header files)

definition section

Global declaration section

main() function section

{

 declaration part

 execution part

}

Sub-program section

function 1

⋮

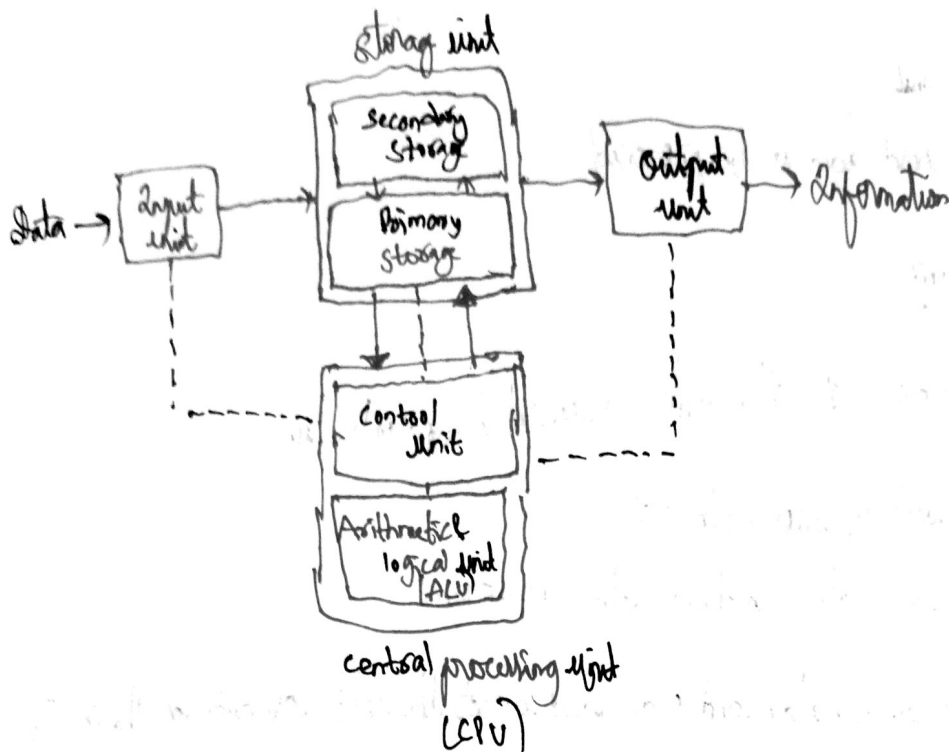
function n

(user defined functions)

- 1) Discussed about computer
- 2) operations
- 3) types of programming languages
- 4) program
- 5) software
- 6) Types of software

22/11/23

5



Input Unit:

- It accepts coded information through electromechanical devices such as keyboard or other sources
- Keyboard is made so that whenever a key is pressed, a corresponding letter or digit is translated into corresponding code & sent to memory for processing

CPU:-

- It is the heart of a computer
- It is the main of the computer
- The main functions are to execute the programs or store the data in memory
- It consists of 3 functional units: Arithmetic & logical unit, control unit & memory unit

Arithmetic & logic unit:-

- It performs arithmetic & logical operations on data

- It controls overall activities of components of computer

Memory unit

- Used to hold data to be processed

Control unit

- The operations of other units are controlled by control unit
- Control unit also helps with I/O
- It also generates synchronisation programs
- It is able to maintain order and direct the entire operation of the system

There are 2 types of memory

1) volatile memory → temporary, wipes when closed

2) non-volatile memory → permanent

Random access memory is an example of volatile memory

Ques

30/8/23

⑦

Structure of C program

1) Documentation section (optional)

- Explains things about the code
- It is optional

Two types: → single line (`//`)
→ multiline (`/* */`)

2) preprocessor section

Ready to use methods are provided by preprocessors

3) Definition section (optional)

define

4) Global declaration (optional)

Place where global variables are declared

5) main()
{ }

Starting point of the program contains statements

6) Sub programs (optional)

Rules of C

- file extension \rightarrow .c
- C is case sensitive
- statements are terminated by semicolon

preprocessor directive

It can be denoted by # include name

printf \rightarrow monitor

scanf \rightarrow keyboard

Q) write a program to print name

include <stdio.h>

void main () {

printf("KTESW");

}

~~Let's do it~~

4/9/23

Lab I

1 9

1) #include <stdio.h>

void main() {

puts("welcome to C programming");

}

2) #include <stdio.h>

void main() {

int dec = 10;

float pi = 3.140000;

char ch = 'p';

printf("%d %f %c", dec, pi, ch);

}

Q) write a program to perform addition of 2 numbers

#include <stdio.h>

int a = 100;

int b = 200;

int c = a + b;

printf("%d", c);

2) write a program to perform addition of 3 numbers

Q) write a program to perform multiplication of 2 numbers

Q) subtract

Q) #include <stdio.h>

void main() {

int a = 30;

int b = 2;

int c = 3;

~~int~~ int Sum = a + b + c;

printf("%d", Sum);

}

Q) #include <stdio.h>

void main() {

int a = 30

int b = 2;

int c = 30 * 2;

printf("%d", c);

Q) #include <stdio.h>

void main() {

int a = 10;

int b = 3;

int c = 10 - 3;

printf("%d", c);

- Q) write a program to find: area of triangle
ii) area of square
iii) area of rectangle

i) #include <stdio.h>

```
void main() {  
    float b = 3;  
    float h = 6;  
    float area = 0.5 * b * h;  
    printf("%d", area);  
}
```

ii) #include <stdio.h>

```
void main() {  
    int a = 3;  
    int b = 4;  
    int ar = a * a;  
    printf("%d", ar);  
}
```

iii) #include <stdio.h>

```
void main() {  
    int a = 5;  
    int b = 2;  
    int ar = a * b;  
    printf("%d", ar);  
}
```

eg) write a program to calculate simple interest

There are 6 steps in program development

- 1) problem definition (understanding the problem)
- 2) problem analysis
- 3) algorithm development
- 4) coding & documentation
- 5) Test & debug
- 6) maintenance

1) problem def:- we should be clear about the purpose of the program and its objectives

2) problem analysis:- After understanding the problem to be solved, we need to identify different ways to solve the problem & evaluate the methods and ch

3) Developing the algorithm:- write the sequential steps of the problem

4) Coding & documentation:- In this, all algorithmic statements to be represented using programming statements of programming languages
or To understand the steps, we need to add comments

5) Test & Debug After completion, we need to tell the program whether we are getting expected output or not finding & fixing errors is debugging

6) Maintenance: The program may be updated with additional features

Algorithms

algorithms can be expressed in any language. They can be defined as sequential program steps in a language

properties of algorithm:-

- finiteness:- It should be terminated after a finite no of steps
- Definiteness:- The program statements need to be stated without any ambiguity
- Input:- There should be zero or more inputs and clearly mentioned
- Output:- 1 or more output
- Effectiveness:- All the steps are sufficiently simple & basic

Algorithm to calculate simple interest

Step 1: Start

Step 2: Input P, T, R

Step 3: Calculate $SI \leftarrow (PTR)/100$

Step 4: print SI

Step 5: stop

5/9/23





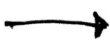

(14)

~~Step 4~~

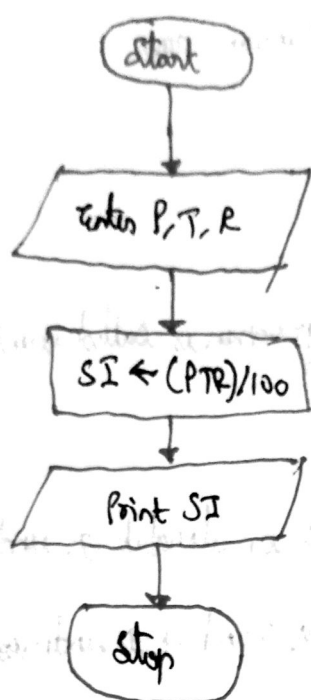
8/

flowchart representation

A program can be represented using flowchart. In flowchart, statements are going to be represented using symbols

Name	Symbol	Description
Oval		used for start & stop
Parallelogram		used for input/output
Rectangle		used for processing/computation
Diamond		used for conditions
Arrow		Denotes logic
Circle		used as connector

flowchart to calculate simple interest



Character set:-

- C language supports all the alphabets from the English language, both lowercase and uppercase
- C supports 10 digits
- C also supports special characters
- white spaces are also supported
eg:- Blank space, Horizontal tab, carriage return, New line

Tokens:-

Smallest unit in a C program

C Token is divided into 6 different types; keywords, operators, constants, special characters & identifier

Keywords:-

They will have predefined meaning in the language. We don't use keywords as variables or identifiers nor with function names.

eg:- auto, double, int

Identifier:-

Name assigned to an element in the program is called identifier

Rules

- 1) The first character must always be an alphabet or underscore
- 2) It should be formed using only letters, numbers or underscore
- 3) A keyword can't be used as identifier
- 4) It should not contain whitespace
- 5) Name must be meaningful

6/9/23

17

Data types:-

Types

various data types are int, float, char etc.
They used to define the type of variable

Operators:-

There are 8 types of operators

- 1) Arithmetic $\rightarrow +, -, /, *, \%$
- 2) Relational
- 3) Logical
- 4) Assignment
- 5) Increment & decrement
- 6) Conditional
- 7) Bitwise
- 8) Special

Expressions \rightarrow collection of operators & operand

Arithmetic

operator	meaning
+	addition
-	subtraction
*	multiplication
/	division
%	modulus

Q) Write a program to implement all arithmetic operators

```
#include <stdio.h>
```

```
*
```

```
void main() {
```

```
int a, b, c;
```

```
a = 10;
```

```
b = 4;
```

```
c = a + b;
```

```
printf("%d", c);
```

```
c = a - b;
```

```
printf("%d", c);
```

```
c = a * b;
```

```
printf("%d", c);
```

```
c = a / b;
```

```
printf("%d", c);
```

```
c = a % b;
```

```
printf("%d", c);
```

Relational operators:-

- Used to compare 2 values
- Binary operators
- Returns true or false

Eg:- <, >, <=, >=, !=, ==

logical and operatorlogical and $\rightarrow \&\&$ logical or $\rightarrow ||$ logical not $\rightarrow !=$

eg:-

a = 10

printf("%d", a)

 $((a > b) || (a > c)) \&\& (a != 20);$

F

F

T

= F

Increment & decrement

Unary is a type of operator that can be used with one operand

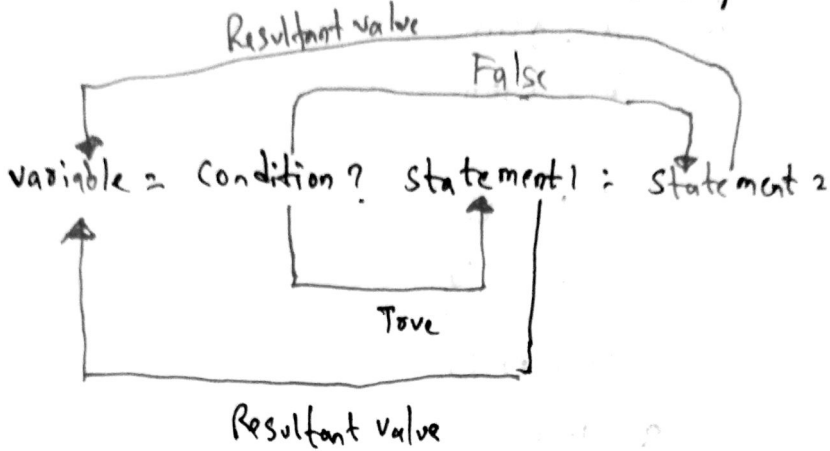
- Post increment:- value will be substituted first, then incremented
 - Pre increment:- value will be substituted first, then decremented
 - Pre increment:- value will be incremented then substituted
 - Pre decrement:- value will be decremented first then substituted
- Increment & decrement operators have higher precedence than other operators
- Postfix increment & decrement have higher precedence than pre increment or decrements

19/9/23

(2)

Conditional operator (ternary operator) :-

Syntax :- (Condition) ? Statement 1 : Statement 2;



Ex:-

```
int a, b, c;
```

```
a = 5;
```

```
b = 6;
```

```
c = (a > b) ? a : b;
```

```
printf("%d", c);
```

Bitwise operators

Bitwise operations are used to perform operations at bit level
also known as bit level programming

Bitwise operations are used in numerical composition for a ~~more~~ faster calculations

operator	Purpose
&	Bitwise and operator
	Bitwise OR operator
^	Bitwise exclusive OR operator
~	One's complement operator
<<	Left shift operator
>>	Right shift operator

Bitwise AND operator (&)

uses boolean algebra, product, denoted by &

$$\text{Eg } a = 0110$$

$$b = 0100$$

$$\begin{array}{r} 0110 \\ 0100 \\ \hline 0100 \end{array}$$

$$= 0100$$

Bitwise OR operator (|)

uses boolean algebra, sum, denoted by |

$$a = 1001$$

$$b = 0100$$

$$\begin{array}{r} 1001 \\ 0100 \\ + \\ \hline 1101 \end{array}$$

Exclusive OR operator (^)

$$\text{XOR} = ^$$

$$1001$$

$$0010$$

$$0101$$

$$0101$$

$$\begin{array}{r} a = 00001100 \\ b = 00001010 \\ \hline 0110 \end{array}$$

Complement operator

~~set~~

left shift operator (\ll)

It substitutes last n number of positions towards left with zeros

$$0101 \ll 2$$

$$\begin{aligned} &0101 \ll 2 \\ &= 010100 \end{aligned}$$

Right shift operator (\gg)

It substitutes last n number of positions towards right with zeros

Q) Write a program to demonstrate bitwise operators;

#include <stdio.h>

void main()

{
int a=10;

int b=6;

printf("%d", a/b);

printf("%d", a&b);

printf("%d", ~a);

printf("%d", a<<2);

printf("%d", b>>2);

}

Data types

Type	Size	range
int	2 bytes	-32,768 to 32,767
float	4 bytes	$1.2E-38$ to $3.4E+38$ (binary)
double	8 bytes	$2.3E-308$ to $1.7E+308$ (10 decimal)
char	1 byte	-128 to 127

Q. 10.

19/9/23

(22)

$$\begin{array}{r} 22 \\ 2 \overline{) 44} \\ 22 \\ \hline 0 \end{array}$$

a) Convert 45 into binary form

$$\begin{array}{r} 2 \overline{) 45} \text{ (1)} \\ 2 \overline{) 22} \text{ 0} \\ 2 \overline{) 11} \text{ (1)} \\ 2 \overline{) 5} \text{ (1)} \\ 2 \overline{) 2} \text{ (1)} \\ \hline 10 \end{array}$$

$$\begin{array}{r} 51 \\ 2 \overline{) 102} \\ 51 \\ \hline 0 \end{array}$$

$$\Rightarrow 101101$$

Binary to decimal

$$\begin{array}{cccccc} 1 & 0 & 1 & 1 & 0 & 1 \\ 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \end{array}$$

$$\Rightarrow 2^0 \times 1 + 2^1 \times 0 + 2^2 \times 1 + 2^3 \times 1 + 2^4 \times 0 + 2^5 \times 1$$

$$= 1 + 0 + 4 + 8 + 0 + 32$$

$$\Rightarrow 1 + 4 + 8 + 32$$

$$\Rightarrow 45$$

a) Convert 110 into binary form

$$\begin{array}{r} 2 \overline{) 110} \\ 2 \overline{) 55} \text{ 0} \\ 2 \overline{) 27} \text{ 1} \\ 2 \overline{) 13} \text{ 1} \\ 2 \overline{) 6} \text{ 1} \\ 2 \overline{) 3} \text{ 0} \\ \hline 21 \text{ 1} \end{array}$$

$$110 \Rightarrow 1101110$$

Q) Convert 128 to ~~hexa~~ Octal & Hexadecimal

$$\begin{array}{r} 8 \overline{) 128} \\ 8 \overline{) 16} - 0 \\ 1 - 0 \end{array}$$

128 \Rightarrow 100 (Octal)

$$\begin{array}{r} 16 \overline{) 128} \\ 8 - 0 \end{array}$$

=

Q) write a program to demonstrate bitwise and, bitwise or operation

A)

```
#include <stdio.h>
void main()
{
    int a = 5;
    int b = 6;

    int c = a & b;
    printf("%d", c);
    printf("%d", a | b);
}
```

$$\begin{array}{r} 2 \overline{) 20} \\ 2 \overline{) 10} 0 \\ 2 \overline{) 5} 0 \\ 2 \overline{) 2} 1 \\ 1 \end{array}$$

$$\begin{array}{r} 2 \overline{) 2} \\ 1 \end{array}$$

Types of expressions

Expression is combination of operators and operands.

There are 4 types:

1) Arithmetic expression ($C \geq a + b$)

2) Conditional expression ($x \% 2 \geq 0$)

3) Logical expressions

4) Relational expressions

Relational Expression

Used to compare 2 operands

Eg:-

$$x \% 2 == 0$$

Logical Expression

We can combine more than one condition

Eg:-

$$(n > 4) \text{ \& } (n < 6)$$

Operators Precedence & Associativity

Operator Precedence \rightarrow Determines which operator performed first in an expression with more than one operator with different precedence

Operators Associativity \rightarrow Used when 2 operators of same precedence appear in an expression

Evaluate the following

$$1) 22 \div 10 \times 5 * 12 - 6 + 2$$

$$= 0$$

$$2) y = 5 + 2 * (7 + 2 - 7) / 3 * (2 - 1 + 3)$$

$$= 5 + 2 * (2) / 3 * (4)$$

$$= 5 + 4 / 12 = \frac{5 + \frac{4}{12}}{1}$$

$$= \frac{5 + \frac{4}{12}}{1} = \frac{5 + \frac{1}{3}}{1} = \frac{16}{3}$$

25/09/23
38/40

25/9/23

(30)

Q) write a program to read a character, an integer & float

A) #include <stdio.h>

```
int main() {
```

```
    int a;
```

```
    float b;
```

```
    char ch;
```

```
    printf("Enter a value");
```

```
    scanf("%d", &a);
```

```
    printf("Enter b value");
```

```
    scanf("%f", &b);
```

```
    printf("Enter a character");
```

```
    scanf("%c", &ch);
```

```
    printf("Char=%c, integer=%d,
```

Q) write a program to read 2 numbers and perform addition

Q) write a program to read 2 float values and perform addition

Q) write a program to read 3 int & apply multiply operation

Q) write a program to perform subtraction from scanf
of 2 nos

Conditional statement

- Conditional statements are used to control the flow of program
- It allows us to control whether a program segment is executed or not
- Evaluates condition or logical expression first and based on its result, the control is transferred to particular statement
- If result is true, then it takes one path else it takes another

Decision making statements in C

One way decision: if

2 way decision: if...else

multi way decision: if...else if...else if...else

Two way decision: ? : (conditional operator)

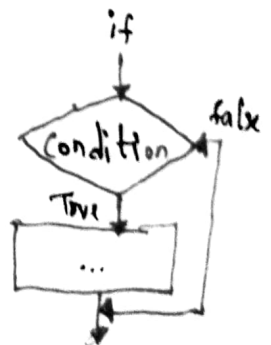
n-way decision: Switch...case

If statement

Syntax: 1) if (Condition) { Statement; }

If is single branch decision making statement

If condition is true then body will be executed



- Q) Write a program to check whether the given number is even & positive or not

```
#include <stdio.h>
```

```
int main () {
```

```
int a;  
printf("enter a value");  
scanf("%d", &a);  
if(a > 0) {  
    printf("The given value is +ve %d", a);  
}  
}
```

Q) write a program to check whether the given number is even or odd

1) Program to check if age is below 18

```
#include <stdio.h>
```

```
int main {
```

```
    int age = 21;
```

```
    if (age < 18) {
```

```
        printf("Below 18"); }
```

```
    else {
```

```
        printf("Above 18"); }
```

```
}
```

2) Program to check if a person belongs to male gender

```
#include <stdio.h>
```

```
int main() {
```

```
    char gen = 'M';
```

```
    if (gen == 'M') {
```

```
        printf("Male"); }
```

```
    else {
```

```
        printf("Not male"); }
```

```
}
```

3) Program to check if number is even or odd

```
#include <stdio.h>
```

```
int main() {
```

```
    int a = 2;
```

```
    if (a % 2 == 0) {
```

```
        printf("even"); }
```

```
    else {
```

```
        printf("odd"); }
```

```
}
```

4) Program to check if this year is 2027

```
#include <stdio.h>
```

```
int main() {
```

```
    int yr = 2023;
```

```
    if (yr == 2027 yr == 2027) {
```

```
        printf("year is 2027");
```

```
    } else {
```

```
        printf("year is not 2027");
```

```
    }
```

5) Program to check if a number is divisible by 7

```
#include <stdio.h>
```

```
int main() {
```

```
    int num = 8;
```

```
    if (num % 7 == 0) {
```

```
        printf("Divisible by 7");
```

```
    } else {
```

```
        printf("Not Divisible by 7");
```

```
    }
```

#include <stdio.h>

int n, s;

printf("Enter a value");

scanf("%d", &n);

r = n % s;

if (r == 0) {

printf("%d is divisible by s", n);

}

Q) Write a program to check if number is divisible by 4210

int n;

printf("Enter a value");

scanf("%d", &n);

if ((n % 10 == 0) && (n % 4 == 0)) {

printf("The number %d is divisible by 4210", n);

}

printf("Program executed");

Q) Write a program whether the no is divisible 2 or 3

#include <stdio.h>

int main() {

int n;

printf("Enter a number");

scanf("%d", &n);

if ((n % 2 == 0) || (n % 3 == 0)) {

printf("Divisible by 2 or 3");

}

Q) write a program to check whether the first number is divisible by second number or not

include <stdio.h>

int main()

{
int n₁, n₂;

printf("Enter first");

scanf("%d", &n₁);

printf("Enter second");

scanf("%d", &n₂);

if (n₁ % n₂ == 0)

{
printf("%d is divided by %d", n₁, n₂);

}

}

Q) write a program to check whether the given number is leap year or not

Q) write a program to check whether the given char is a or not

char ch;

printf("Enter a char");

scanf("%c", &ch);

if (ch == 'a')

{
printf("%c is a");

}

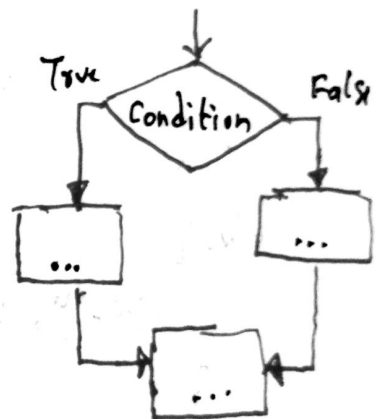
Q) Write a program to read 2 characters and check if they are equal or not

```
char ch1, ch2;
printf("Enter a char");
scanf("%c", &ch1, &ch2);
if (ch1 == ch2) {
    printf("they are equal");
}
```

If....else:-

Syntax:

```
if (condition) {
    statements;
}
else {
    statement 2;
}
```



If the condition is true, if block gets executed

If the condition is false, else block gets executed

Q) Write a program to check whether given no is even or odd

```
int n;
printf("Enter n");
scanf("%d", &n);
if (n % 2 == 0) {
```



```

    printf("The no. is even", n);
}
else {
    printf("The no. is odd", n);
}

```

6) write a program to check whether the given char is Vowel

```

char ch;
printf("Enter a character");
scanf("%c", &ch);
if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {
    printf("Char is vowel");
}
else {
    printf("not vowel");
}

```

7) write a program to read 3 subject marks, find total avg & grade

27/9/23

39

Q) WAP to print odd or even number

```
#include <stdio.h>
```

```
void main() {
```

```
    int a;
```

```
    printf("Enter Number: ");
```

```
    scanf("%d", &a);
```

```
    if (a % 2 == 0) {
```

```
        printf("Even number"); }
```

```
    if (a % 2 != 0) {
```

```
        printf("Odd number"); }
```

```
}
```

Q) write a program to check whether a student is pass or fail by taking 3 subject

```
#include <stdio.h>
```

```
int main() {
```

```
    int m1, m2, m3;
```

```
    printf("Enter 3 subjects");
```

```
    scanf("%d %d %d", &m1, &m2, &m3);
```

```
    if (m1 > 35 && m2 > 35 && m3 > 35) {
```

```
        printf("result is pass");
```

```
    }
```

```
    else {
```

```
        printf("fail"); }
```

40
Q1) WAP to print positive or negative no using if else

```
#include <stdio.h>
```

```
void main() {
```

```
    int a;
```

```
    printf("Enter Numbers");
```

```
    scanf("%d", &a);
```

```
    if (a > 0) {
```

```
        printf("Positive no");
```

```
    } else {
```

```
        printf("Negative no");
```

```
    }
```

Q2) WAP to find largest no from given 2 nos using if

```
#include <stdio.h>
```

```
void main() {
```

```
    int a, b;
```

```
    printf("Enter two numbers");
```

```
    scanf("%d %d", &a, &b);
```

```
    if (a > b) {
```

```
        printf("%d is largest", a);
```

```
    }
```

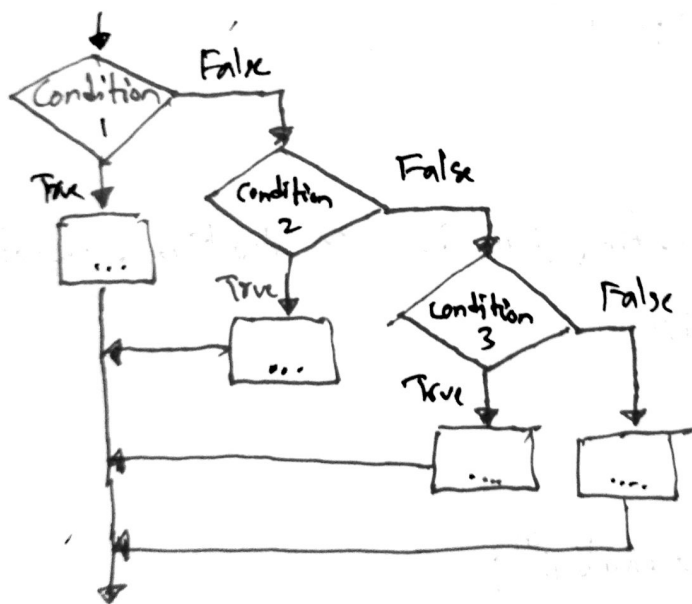
```
    if (a < b) {
```

```
        printf("%d is largest", b);
```

```
    }
```

if...else if...else if...else ladder

- If else if...else if...else is multi branch decision making statement.
- If first if condition is true then remaining if conditions will not be evaluated
- If first if condition is false then second if condition will be evaluated and if it is true, then remaining if conditions will not be evaluated
- If...else if...else if...else is also known as if...elseif ladder



3-10-23

Q) write a program to demonstrate if else

A)

```
int m1, m2, m3, avg;  
printf("Enter 3 ");  
scanf("%d%d%d", &m1, &m2, &m3);  
avg = (m1 + m2 + m3) / 3;  
if (avg >= 80)  
    printf("Grade-A");  
else if (avg >= 70 && avg < 80)  
    printf("Dist");  
else if (avg < 60 && avg < 70)  
    printf("First");  
elseif (
```

Q) write a program to find if no is zero, positive or negative

```
#include <stdio.h>  
void main() {  
    int a;  
    printf("Enter Number:");  
    scanf("%d", &a);  
    if (a > 0)  
        printf("tve No");  
    else if (a == 0)  
        printf("zero");  
    else  
        printf("negative no");  
}
```

Nested if

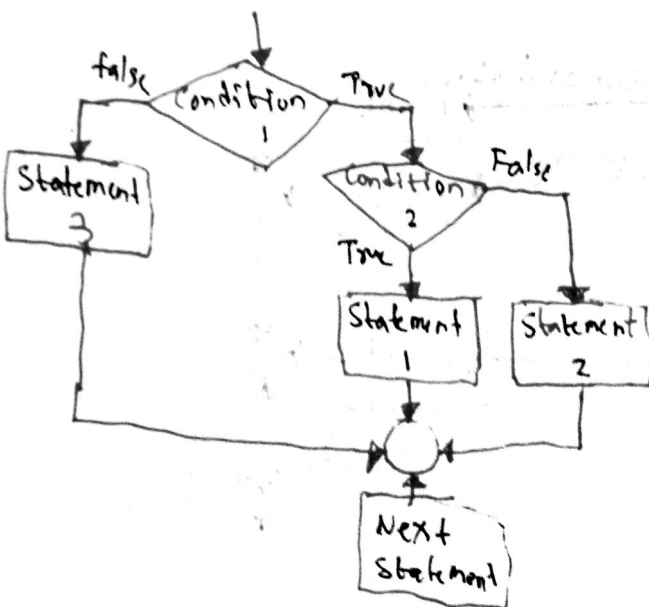
- If condition-1 is satisfied then condition 2 will be evaluated. If it is true, statement-1 will be executed.
- If condition-1 is false then Statement-3 will be executed.

format

```

if (condition-1)
{
    if (condition-2) {
        statement-1;
    }
    else {
        statement-2;
    }
}
else {
    statement-3;
}

```

Flowchart

4/10/23

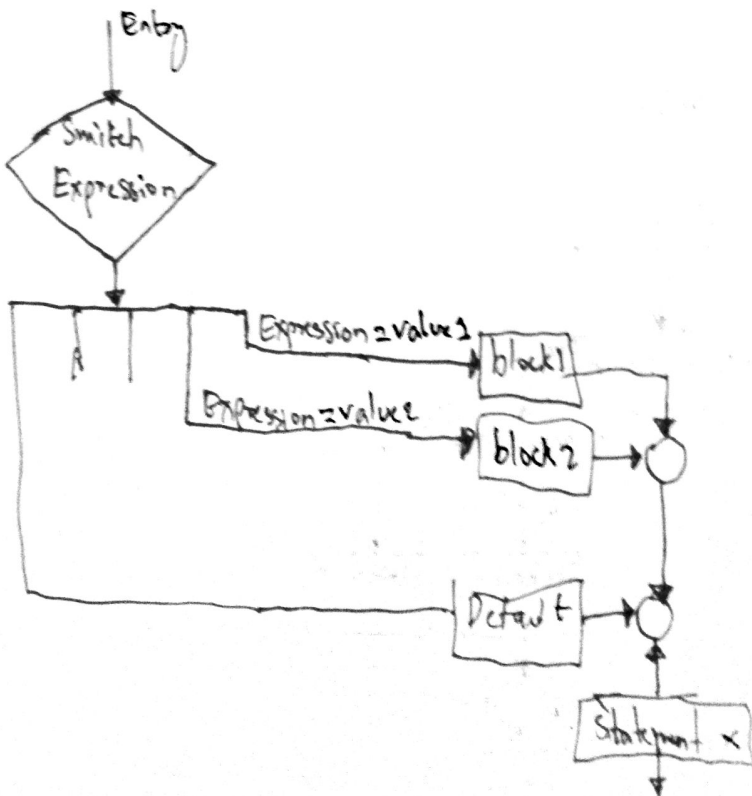
Switch statement

Syntax:

```
Switch (expression) {  
    case value-1:  
        block-1;  
        break;  
    .....  
    .....  
    default:  
        default-block;  
        break;  
}
```

Statement ->

Flowchart



Properties:

- 1) Contains an expression, takes it as input
- 2) Various cases are listed
- 3) Default block is present with default case
- 4) Break statement ends the case

(29)
Q1) write a program to print the given number in words:

A) int a;

printf("Enter a no");

scanf("%d",&a);

switch(a) {

Case 1:

printf("one");

break;

Case 2:

printf("two");

break;

Case 3:

printf("three");

break;

Case 4:

printf("four");

break;

Case 5:

printf("five");

break;

} default:

printf("is valid no");

break;

}

Q7) write a program to check if a given character is vowel using switch

```
A) char ch;  
printf("Enter char");  
scanf("%c", &ch);
```

```
switch(ch){
```

```
    case 'a':
```

```
        printf("Vowel");
```

```
        break;
```

```
    case 'e':
```

```
        printf("Vowel");
```

```
        break;
```

```
    case 'i':
```

```
        printf("Vowel");
```

```
        break;
```

```
    case 'o':
```

```
        printf("Vowel");
```

```
        break;
```

```
    case 'u':
```

```
        printf("Vowel");
```

```
        break;
```

```
    default:
```

```
        printf("Not vowel");
```

```
        break;
```

```
}
```

a) Write a program to implement simple arithmetic calculator using switch & case

```
A) int a, b;  
    char ch;   
    printf("a b");  
    scanf("%d %d", &a, &b);  
    printf("Enter operator");  
    scanf("%c", &ch);  
    switch(ch) {  
        case '+':  
            printf("Add = %d", (a+b));  
            break;  
        case '-':  
            printf("Sub = %d", (a-b));  
            break;  
        case '*':  
            printf("Pro = %d", (a*b));  
            break;  
        case '/':  
            printf("div = %d", (a/b));  
            break;  
        case '%':  
            printf("mod = %d", a%b);  
            break;  
        default:  
            printf("invalid operator");  
            break;  
    }  
}
```

Rules

- Switch expression must be integral type
- case labels need to be unique
- Case labels must be constants
- break is a ~~must~~ optional

Loop

9/10/23

49

Loops:-

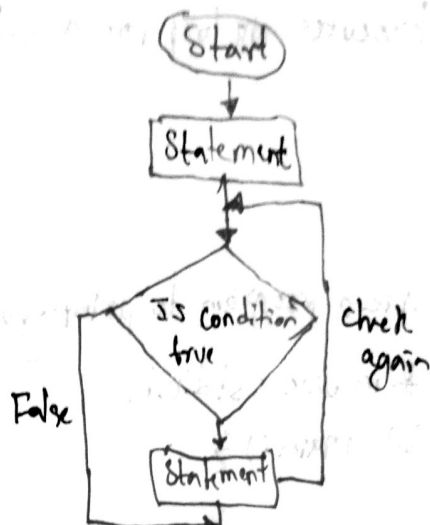
There are 3 types of loops in C

- 1) while
- 2) do... while
- 3) for

While:-

Syntax

```
while (Expression/condition) {  
    Statement 1;  
    Statement 2; }  
False
```



- Used as a loop

- Executes the given block of code until the condition no longer

satisfies

- Entry condition loop

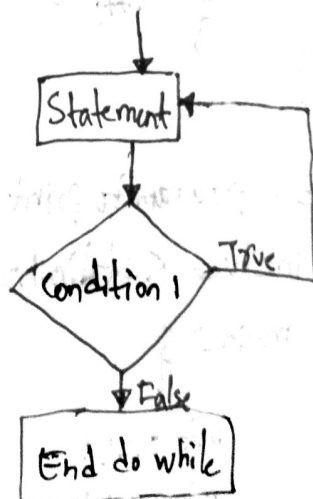
do... while:-

Syntax:-

```
do {
```

```
    Statement 1;
```

```
} while (condition);
```



- Exit condition loop

- Executes the do block until condition no longer satisfies

- Executes once even if condition is not true

for loop:-

```
for(i=v; condition; increment){  
    Statement;  
}
```

- Executes the loop for a selected no of times

Q) write a program to print your name 10 times

```
#include <stdio.h>
```

```
int main() {
```

```
    int i;
```

```
    i=0;
```

```
    while(i<10){
```

```
        printf("my name is kits");  
        i++;
```

```
    }
```

Q) write a program to print 1-10 numbers

```
#include <stdio.h>
```

```
int main() {
```

```
    int x=1;
```

```
    int i=0;
```

```
    while(i<=10){
```

```
        printf("%d", x);
```

```
        x++;
```

```
    }
```

Q7) write a program to perform addition of 1-10

```
#include <stdio.h>
```

```
int main() {
```

```
    int i = 1;
    int s = 0;
    while (i <= 10) {
```

```
        s = s + i
```

```
        i++;
    }
```

```
    printf("%d", s);
```

```
}
```

Q8) write a program to print 1 to given number

```
#include <stdio.h>
```

```
int main() {
```

```
    int i = 1;
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    while (i <= n) {
```

```
        printf("%d", i);
```

```
        i++;
```

```
}
```

Q) write a program to find Sum of 1 - given no
`#include <stdio.h>`

```
int main(){
```

```
    int i = 1
```

```
    int s = 0
```

```
    int n;
```

```
    scanf("%d", &n);
```

```
    while (i <= n) {
```

```
        s = s + i
```

```
        i++;
```

```
    }
```

```
    printf("%d", s);
```

```
    printf("%d", s);
```

Q) write a program to print numbers b/w 2 given values

```
int i, n1, n2;
```

```
printf("enter no");
```

```
scanf("%d %d", &n1, &n2);
```

```
while (n1 <= n2) {
```

```
    printf("%d", n1);
```

```
    n1++;
```

```
}
```

Q) write a program to print even numbers in a given range

a) write a program to print odd nos in a given range

a) write a program to find factorial of a given number

n) int i, fact, n;

scanf("%d", &i);

fact = 1;

i = 1;

while (i <= n) {

fact = fact * i;

i++;

}

printf("factorial of %d = %d", n, fact);

}

a) write a program to find factorial using for & do while

int i, fact;

fact = 1

scanf("%d", &n);

i = 1;

do {

fact = fact * i;

i++;

} while (i <= n)

printf("%d", fact)

10/10/23
Q) write a program to reverse a number

```
int rev, rem, n;
```

```
rev = 0;
```

```
scanf("%d", &n);
```

```
while(n != 0) {
```

```
    rem = n % 10;
```

```
    rev = rev * 10 + rem;
```

```
    n = n / 10;
```

```
}
```

```
printf("Reverse of %d = %d", n, rev)
```

```
}
```

Q) write a program to print a palindrome or not

```
int rev, n, old;
```

```
rev = 0;
```

```
scanf("%d", &n);
```

```
old = n;
```

```
while(n != 0) {
```

```
    rem = n % 10;
```

```
    rev = rev * 10 + rem;
```

```
    n = n / 10;
```

```
}
```

```
if (old == rev) {
```

```
    printf("Palindrome");
```

```
else {
```

```
    printf("Not Palindrome");
```

```
}
```

(2) write a program to print armstrong number

```
int n, old, rem, arm;
```

```
arm = 0;
```

```
printf("Enter n");
```

```
scanf("%d", &n);
```

```
old = n;
```

```
while (n != 0) {
```

```
    rem = n % 10
```

```
    arm = arm * rem + rem * rem * rem;
```

```
    n = n / 10
```

```
}
```

```
if (old == arm) {
```

```
    printf("%d is armstrong", old);
```

```
else {
```

```
    printf("not armstrong");
```

Break:-

- Used to break the loop
- makes an early exit from block
- Can be used in control statements

continue

- Helps in avoiding remaining statements in a current iteration of loop & continuing next iteration
- loop constructs only

Q) write a program to print even numbers in b/w 1-10 using continue

```

i = 0
while (i < 10) {
    i++;
    if (i % 2 == 1) {
        continue;
    }
    else {
        printf("%d", i);
    }
}

```

Q) write a program to print 1-100 nos w/o printing multiples of 3

11/10/25

(6)

Nested loops

• It is a loop within a loop

Fig 1

```
for (i=0; i<=4; i++) {  
    for (j=0; j<=4; j++) {  
        printf("%d", j);  
    }  
}
```

Q) write a program to print

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

```
int i, j;  
for (i=0; i<=4; i++) {  
    for (j=0; j<=4; j++) {  
        printf("%d", i);  
        printf("\n");  
    }  
}
```

6) write a program to Print

1	2	3	4	5
6	7	8	9	10

```
int i, j, k;
```

```
int k = 1;
```

```
for (i = 0; i <= 4; i++) {
```

```
    for (j = 0; j <= 4; j++) {
```

```
        printf("%d", k);
```

```
        k++;
```

```
    }
```

```
}
```

2) write a program to Print

1
2 2
3 3 3
4 4 4 4

```
int i, j, k = 1;
```

```
for (i = 1; i <= 4; i++) {
```

```
    for (j = 1; j <= i; j++) {
```

```
        printf("%d", i);
```

```
    }
```

```
}
```

Q write a program

Execution of nested for loops

for (initialization; condition; increment/decrement) {

for (initialization; condition; increment/decrement) {
Statement;

}

}

Arrays:-

A single variable consisting of more than one values

Syntax:-

datatype arrayname [Size]

Arrays have indexes and they start from 0

It ~~can~~ contains values of the same data types

Advantages:-

- 1) Code optimising
- 2) Ease of traversing
- 3) Ease of sorting
- 4) Random Access

Disadvantages:-

- 1) Fixed size

Array Initialization

- Array ~~must~~ ~~has to~~ can be initialised at the time of declaration

```
int marks[5] = {1, 2, 3, 4, 5}
```

```
int i;
```

```
Print ("Array values");
```

```
for (i=0; i<5; i++)
```

```
printf ("%d", marks[i]);
```

2) write a program to read n values and display by using arrays

```
int a[10];
```

```
int n;
```

```
int i = 0;
```

```
Print f ("enter n values");
```

```
scanf ("%d", &n);
```

```
printf ("enter values");
```

```
for (i=0; i<n; i++) {
```

```
scanf ("%d", &a[i]);
```

```
}
```


30/10/23

Types of arrays

- i) one dimensional array $\rightarrow A_i[i]$
- ii) two dimensional array $\rightarrow A_i[i][j]$
- iii) ~~4th~~ Multi dimensional array

1D:-

Syntax

`Datatype arr.name[size]`

2D array:-

Syntax

`Datatype arr.name[row][col]`

It is a matrix representation of data and the data can be accessed via row & col indices

Q) write a program to read and display a double dimension integer array

A) `#include <stdio.h>`

`int main()`

`{`
`int i, j;`

`int a[3][3];`

62

```
printf("enter matrix values:");
```

```
for(i=0; i<3; i++){
```

```
    for(j=0; j<3; j++){
```

```
        scanf("%d", &a[i][j]);
```

```
    }
```

```
}
```

```
printf("matrix ");
```

```
for(i=0; i<3; i++){
```

```
    for(j=0; j<3; j++){
```

```
        printf("%d", a[i][j]);
```

```
    }  
    printf("\n");
```

```
}
```

Q) write a program to read $m \times n$ matrix and display in matrix format

a) program on if statement

a) one program on switch case

a) one program on while loop

a) one program on for loop

a) one program on do while

31/10/23

(68)

```
#include <stdio.h>
```

```
int main() {
```

```
    int i, j;
```

```
    int a[10][10]
```

```
    int a[m][n];
```

```
    scanf("%d", &a[m])
```

```
    int m, n
```

```
    scanf("%d %d", &m, &n);
```

```
    int a[m][n];
```

```
    printf("Enter matrix values");
```

```
    for(i=0; i<m; i++) {
```

```
        for(j=0; j<n; j++) {
```

```
            scanf("%d", &a[i][j]);
```

```
        }
```

```
    }
```

```
    for(i=0; i<m; i++) {
```

```
        for(j=0; j<n; j++) {
```

```
            scanf printf("%d", a[i][j]);
```

```
        }  
        printf("\n");
```

```
    }
```

Addition of matrix:-

```
#include <stdio.h>
```

```
int main() {
```

```
    int a[10][10], b[10][10], c[10][10];
```

```
    int m, n, p, q;
```

```
    printf("Enter m x n");
```

```
    scanf("%d %d", &m, &n);
```

```
    printf("Enter p x q");
```

```
    scanf("%d %d", &p, &q);
```

```
    if (m != p || n != q) {
```

```
        printf("Not possible");
```

```
    }
```

```
    else {
```

```
        printf("Enter matrix");
```

```
        for (i = 0; i < m; i++) {
```

```
            for (j = 0; j < n; j++) {
```

```
                scanf("%d", &a[i][j]);
```

```
            }
```

```
        }
```

```
        for (i = 0; i < m; i++) {
```

```
            for (j = 0; j < n; j++) {
```

```
                scanf("%d", &b[i][j]);
```

```
            }
```

```
for (i = 0; i < m; i++) {
```

```
    for (j = 0; j < n; j++) {
```

```
        c[i][j] = a[i][j] + b[i][j]
```

```
    }    printf("%d", c[i][j]);
```

```
    }    printf("\n");
```

Print

2D Array Initialisation

2D Arrays can be declared using

```
int disp[2][4] = { {10, 11, 12, 13}, {14, 15, 16, 17} };
```

or

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17 }
```

Columns are important for deinitialization

a) Matrix multiplication

b) Trace of a matrix

c) Transpose of a matrix

2) matrix multiplication

```
#include <stdio.h>
```

```
int main()
```

```
{  
    int i, j, m, n, p, q, r, s;
```

```
    int a[10][10], b[10][10], c[10][10];
```

```
    printf("Enter m x n");
```

```
    scanf("%d %d", &m, &n);
```

```
    printf("Enter p x q");
```

```
    scanf("%d %d", &p, &q);
```

```
    printf("Enter r x s");
```

```
    scanf("%d %d", &r, &s);
```

```
    if (m != p || n != q) {
```

```
        printf("Not a square matrix");
```

```
    }
```

```
    else {
```

```
        printf("Enter a matrix");
```

```
        for (i = 0; i < m; i++) {
```

```
            for (j = 0; j < n; j++) {
```

```
                printf("Enter m matrix");
```

```
                scanf("%d", &a[i][j]);
```

```
            }
```

```
        }
```

```
        for (i = 0; i < m; i++) {
```

```
            for (j = 0; j < n; j++) {
```

```
                printf("Enter p matrix");
```

```
                scanf("%d", &b[i][j]);
```

```
            }
```

```
        }
```

```

printf("Product:");
for (i=0; i<m; i++) {
    for (j=0; j<n; j++) {
        c[i][j] = a[i][j] * b[i][j];
        printf("%d", c[i][j]);
    }
    printf("\n");
}
}
}
}

```

Q) Transpose of a matrix

```
#include <stdio.h>
```

```
int main() {
```

```
    int i, j, m, n, p, q;
```

```
    int a[10][10], b[10][10];
```

```
    printf("Enter m x n");
```

```
    scanf("%d %d", &m, &n);
```

```
    printf("Enter the matrix n");
```

```
    for (i=0; i<m; i++) {
```

```
        for (j=0; j<n; j++) {
```

```
            scanf("%d", &a[i][j]);
```

```
        }
```

```
    }
```

```
printf("Entered matrix is \n");
```

```
for (i=0; i<m; i++) {
```

```
    for (j=0; j<n; j++) {
```

```
        printf("%d", a[i][j]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
printf("Transpose of matrix: \n");
```

```
for (i=0; i<m; i++) {
```

```
    for (j=0; j<n; j++) {
```

```
        b[j][i] = a[i][j];
```

```
        printf("%d", b[j][i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
}
```


2/11/23

(74)

Searching algorithms

Linear Search:

- In this type, we search all the elements of an array one by one until we find the suitable element if not we return null
- It is simplest searching algorithm
- Used widely in unordered lists

Time complexity = $O(n)$

The array

Ex:-

Let array be

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

Let searching element be $x = 41$

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

↑
 $x \neq 70$

Continues until 41 is found

(75)

0	1	2	3	4	5	6	7	8
70	40	30	11	57	41	25	14	52

↓
K=41

when $K=41$, ~~array~~ algorithm returns the element's index that matched

Ex:-

#include <stdio.h>

int main() {

int array[100], search, n;

printf("Enter no of elements");

scanf("%d", &n);

printf("Enter %d integers", n);

for (c=0; c<n; c++) {

scanf("%d", &array[c]); }

printf("Enter a no to search);

scanf("%d", &search);

for (c=0; c<n; c++) {

if (array[c] == search) {

printf("%d is at %d index", search, c);

break;

}

}

(76)

```
if (c == n) {
```

```
    printf("%d is not present in array", search);
```

```
}
```

```
}
```

6/11/23

Unit-3

(77)

Strings

- Group of characters ending with null character
- `char ch[] = {"a e i o u \0"};`
- Without a null character it is called character array
- String is a one dimensional array of characters
- Eg:- `char str[] = "Home";`

Index →	0	1	2	3	4
str →	H	O	M	E	\0

- String HOME contains 5 characters including null character which is automatically added by the compiler
- We cannot ~~assign~~ use assignment operator with strings

Reading Strings

- We can use `scanf` with `%s` to read a string
- But it terminates its input on the first white space it encounters

Eg:- `#include <stdio.h>`

`int main()`

`char st[20];`

`printf("Enter a string");`

`scanf("%s", st);`

`printf("String: %s", st);`

`}`

Output:

enter a string: KITS warangal

string = KITS

If we insert more char than the defined one, we get errors

Method 2 using edit set (conversion code %[-]) we can read whitespaces
Eg:- `scanf("%[-\n]", line)`

Method 3 we can use `getchar()` & `gets()` to read whitespaces

Eg:- `char ch; int c=0; char line[80]`

do {

`ch = getchar();`

`line[c] = ch;`

`c++;`

{

`while (ch != '\n');`

`c = c - 1;`

`line[c] = '\0';`

7/11/23

(79)

- a) write a program to demonstrate gets()
- b) write a program to read your name and address and display

a) #include <stdio.h>

int main()

~~int~~ char ch[60];

printf("Enter something");

~~gets(ch[0])~~

gets(ch);

b) #include <stdio.h>

int main()

char name[60];

printf("Enter name");

scanf("%[^\n]", name);

char add[100];

printf("Enter address");

scanf("%[^\n]", add);

printf("Name & address: %s & %s", name, add);

Writing strings

We can use following to print strings:

① printf

② putchar()

Eg:- `char ch = 'K';`
`putchar(ch);`

We can only print one character at once w/ `putchar()`

③ puts()

It is similar to `gets()`

Q) write a program to demonstrate `putchar`

```
char name[6] = "Kishu";  
int i = 0;  
printf("the text is");  
for(i = 0; i < 6; i++)  
    putchar(name[i]);  
}
```

(81)

Q) write a program to find length of a string

```
char name[30];
```

```
int count, i=0;
```

```
printf("Enter name");
```

```
scanf("%s", &name);
```

```
for (i=0; name[i] != '\0'; i++) {
```

```
    count
```

```
    count++;
```

```
}
```

```
printf("length of name = %d", count);
```

Q) ~~write a program~~

functions we can perform on string

1) Counting → length

2) Copy -

3) Compare

4) Concatenate

5) reverse a string

8/11/23

String handling functions

- Included in `string.h`
- `strcpy()` → Copies string 2 to string 1
- `strncpy(string1, string2, s)` → Copies first s characters in string 2 to string 1
- `strlen(string1)` → Returns total number of chars
- `strcat(string1, string2)` → appends string 2 to string 1
- ~~strncat~~
- `strncat(string1, string2, 4)` → appends first 4 chars of string 2 to 1
- `strcmp(string1, string2)` → Returns 0 if string 1 & string 2 are same
- `strncmp(string1, string2, 4)` → compares first 4 chars of string 1 & string 2
- `strcmp(string1, string2)` → compares 2 strings
- `strlwr(string1)` → Converts all chars of string 1 to lower case
- `strupr(string1)` → Converts all chars of string 1 to upper case
- `strrev(string1)` → It reverses value of string 1

Implementation of stolen

Eg:- ~~char s~~
 char str[] = "wastk";
 k = stolen(str);

Implementation of strcpy()

char str1[] = "Hello";
 char str2[] = "World";
 strcpy(str2, str1);

Implementation of strncpy()

char str1[] = "Hello";
 char str2[] = "World";
 strncpy(str2, str1, 4);

Implementation of strcat()

char str1[] = "Hello";
 char str2[] = "World";
 strcat(str1, str2);

Implementation of strncpy()

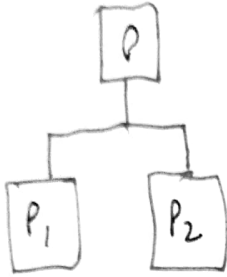
char str1[] = "Hello";
 char str2[] = "World";
 strncpy(str1, str2, 4);

15/11/23

(89)

Functions :- group of statements to perform a task

① Modularity \rightarrow process of dividing big problem into small problem



② Code reusability \rightarrow using of previously written code

③ Functions can be called for any number of times

We can use functions to divide programs into smaller parts

By using functions we can achieve modularity, code reusability

with the help of functions, we can implement programs easily

we can debug easily

we can modify the code easily

Syntax:

```

datatype functionname() {
    // statement;
}

```

Built in functions

Some functions are provided by 'C' by default

Eg: printf(), scanf(), getchar(), puts(), getch(),
getch()

User defined functions

functions implemented by users

Eg: legpiece(), chickenwing()

C provides mechanism to write and to ~~use~~ call functions

Functions in C can be implemented in 3 steps

- 1) Function declaration
- 2) Function definition
- 3) Function invocation

Function declaration

It contains name of the function, return type & arguments

Function definition

It contains actual code of the function

Function calling

Function calling is a statement is that execution of a function

Q) Write a program to perform addition of 2 numbers

```
void add()
```

```
    sum = a + b;
```

```
    printf("sum = %d", sum);
```

```
}
```

```
int main()
```

```
    a = 10;
```

```
    b = 20;
```

```
    add(a, b);
```

```
}
```

Q) Program to print something 3 times

```
void legsprice() {
```

```
    printf("Cost: %d", legs);
```

```
    #printf("Cost: %d", legs);
```

```
    printf("Cost: %d", legs)
```

```
}
```

```
int main() {
```

```
    int legs = 300;
```

```
    legsprice(300 legs);
```

```
}
```

Types

1) Function w/ no argument & no return

2) Function w/ no argument & returns value

3) Function with argument but no return value

4) Function w/ argument & returns value

14/11/23

Actual parameters

parameters which we pass to function

Formal parameters

variables

Parameters which we use to declare a function

Q1 Write a program to demonstrate formal parameters & Actual parameters

~~int a~~

void addition(int m, int n)

{
int c;

c = m + n;

printf("Addition = %d", c);

}

int main()

{
int x, y;

printf("Enter 2 values calling first time");

x = 10; y = 20;

printf("Enter 2 values");

scanf("%d %d", &x, &y);

Scope

Accessible area of function

Eg:

```
#include <stdio.h>
```

```
void addNumbers();
```

```
int main() {
```

```
    addNumbers();
```

```
    return 0;
```

```
}
```

```
void AddNumbers() {
```

```
    int num1, num2, sum;
```

```
    scanf("%d", &num1);
```

```
    scanf("%d", &num2);
```

```
    sum = num1 + num2;
```

```
    printf("Sum: %d\n", sum);
```

```
}
```

Q/wr to find factorial ~~using~~ using various user defined functions

21/11/23

Recursion

Calling a function inside function

```
Eg: void fun() {  
    fun();  
}
```

2 types of recursion

- ① Direct recursion → calling same function
- ② Indirect recursion → calling function in another function

Properties

1) Can be infinite

2) Base Case

→ It allows recursion to stop

3) Progressive approach

→ A recursive algorithm must change its state in such a way that it moves forward to the base case

Q) WAP to find factorial using recursion

```
#include <stdio.h>
int fact(int);
void main() {
    int n, f;
    scanf("%d", &n);
    printf("factorial = %d", f);
}
```

```
int fact(int n) {
    if (n == 0) {
        return 1;
    }
    else if (n == 1) {
        return 1;
    }
    else {
        return n * fact(n-1);
    }
}
```

Q) WAP to implement for string palindrome

```
Q) #include <stdio.h>
#include <string.h>
```

```
int main() {
    char str[100];
    int l, h, flag = 1;
    printf("Enter the string: ");
    scanf("%s", str);
    l = 0;
    h = strlen(str) - 1;
```

```

while (l < h) {
    if (str[l] != str[h]) {
        flag = 0;
        break;
    }
    l++;
    h--;
}

if (flag == 1) {
    printf("%s is palindrome\n", str);
}
else {
    printf("%s is not palindrome\n", str);
}

return 0;
}

```

Q) palindrome using function

```

#include <stdio.h>
#include <string.h>

int main() {
    char str[100], rev[100], int i;
    printf("Enter string to check: ");
    scanf("%s", str);
    strcpy(rev, str);
    reverse(rev);
}

```

if (strcmp(str, rev) == 0) {

printf("%s is palindrome", str);

else

printf("%s is not palindrome", str);

28/1/23

We can use 2 types of variables in functions

- ① local variable → declared inside function
- ② global variable → declared outside function can be accessed by any variable

scope → limit of accessibility

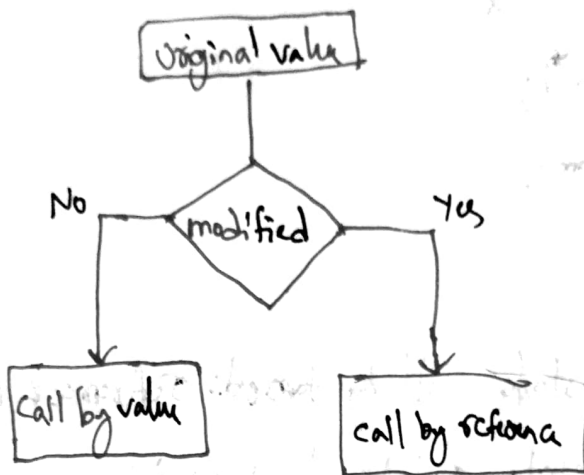
lifetime of a ~~function~~ local variable is limited to the function

Eg:-

```
int a; b;  
void add (int k, int j);  
int main() {  
    x = 10;  
    y = 20;  
    add(x, y);  
    a = x + y;  
    printf("%d", a);  
}  
void add (int x, int y) {  
    int z;  
    z = x + y;  
    printf("%d", z);  
}  
q. a = 10 + 20  
    printf("%d", a);  
}
```

There are 2 types of functions based on the type of arguments we are passing

- i) call by value
- ii) call by reference



Call by value Eg: ✓

void f, (int x);

Call by reference Eg:-

void f(int *x);

Write a program to swap 2 numbers using call by reference

```
#include <stdio.h>
```

```
void swap(int *a, int *b);
```

```
int main(){
```

```
    int m=22, n=44
```

```
    printf("values b/f swap m=%d & n=%d", m, n);
```

```
    swap(&m, &n);
```

(16)

```
printf("values of swap are %d & %d, m, n);
```

```
void swap(int *a, int *b){
```

```
    int tmp;
```

```
    tmp = *a;
```

```
    *a = *b;
```

```
    *b = tmp;
```

```
}
```

Q) WAP to demonstrate call by ~~reference~~: reference call by value

Q) WAP to demonstrate global & local variable

29/11/23

Unit-9

97

User defined datatypes (Structs)

used to let the user define their own data type
structure is collection of different datatypes

Structures can be defined as user defined datatypes
we use 'struct' for creating structures

Information

```
struct tag/name {
```

```
    type name1;
```

```
    type name2;
```

```
};
```

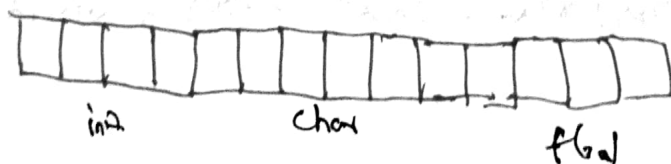
Ex: struct employee {

int id;

char name[25];

float salary;

};



2 ways to declare struct

- 1) By struct keyword with in main()
- 2) Declaring variable with structures declaration

①

```
struct employee {
    int id;
    char name[50];
    float salary;
};
```

struct employee e1, e2;

②

```
struct employee {
    int id;
    char name[50];
    float salary;
};
```

We can access members of struct in 2 ways

- i) By using dot operator (.)
- ii) By using structure pointer operator (→)

2) WAP to demonstrate a structure

a) WAP to create employee record with the following fields, empno, empname, salary & read 2 record values & display max salary

1/12/23

99

Q) write to create a struct with following fields
1) student ~~address~~, roll no, name, total marks

```
#include <stdio.h>
```

```
struct student {
```

```
    int roll no;
```

```
    char name[30];
```

```
    int total_marks;
```

```
} stu;
```

```
int main() {
```

```
    printf(" ");
```

04/11/22

100

Q) Write a p to demonstrate array of structures

Ans

A) Struct employee

```
char empname[20];  
int empid;  
float esalary;  
}
```

```
int main() {
```

```
    struct employee emp[10];
```

```
    int n;
```

```
    printf("Enter n value");
```

```
    scanf("%d", &n);
```

```
    printf("Details");
```

```
    for(i=0; i<n; i++) {
```

```
        printf("Enter name id, sal");
```

```
        scanf("%s %d %f", &emp[i].empname, &emp[i].empid,  
              &emp[i].esalary);
```

```
    }
```

```
    printf("n details are /n");
```

```
    for(i=0; i<n; i++) {
```

```
        printf("name = %s, id = %d, salary = %d", emp[i].empname,  
              &emp[i].empid,  
              &emp[i].esalary);
```

```
    }
```

Q) write to find total & average of 5 student records
fields are student name, roll no, 3 subject marks, total & average

Nested Structure

Structure inside a structure

Eg

```
struct address {
```

```
    char city[20];
```

```
    char hno[10];
```

```
    int pin;
```

```
    char streetName[14];
```

```
};
```

```
struct employee {
```

```
    int id;
```

```
    char name[20];
```

```
    struct address add;
```

```
};
```

```
void main() {
```

```
    struct employee emp;
```

```
    printf("Enter employee information: \n");
```

```
    scanf("%d%s%s%d%s", &emp.id, &emp.name, &emp.add.city, &emp.add.pin, &emp.add.streetName);
```

```
    printf("Printing employee information...\n");
```

```
printf ("Name: %s\n City: %s\n Pincode: %d\n",
        emp.name, emp.add.city, emp.add.pincode,
        emp.add.streetName);
```

needed
2 types of structures:

- 1) Separate structure
- 2) Embedded structure

Embedded structure

It lets us declare struct inside struct

Eg:-

```
struct Employee {
    int id;
    char name[20];
    struct Date {
        int dd;
        int mm;
        int yyyy;
    };
};
```

Union:

used to create structures like types but limited in size
only memory is allocated to max size variable

Ex: Union std {

int m;
char str[40];
}

memory allocated to variable with max size

Example 1:
int m; char str[40];

int m; char str[40];
int m; char str[40];
int m; char str[40];
int m; char str[40];
int m; char str[40];
int m; char str[40];
int m; char str[40];
int m; char str[40];

if both variables are used in a program, then the size of the program will be large

c/12/4/23

Pointer Arithmetic

Pointers \rightarrow address data types

Store address of another variable

Doesn't store any value but only addresses

a) Syntax

```
int *name = 2;
```

Eg:-

```
int *pt2, *pt1;
```

```
int a;
```

```
int b, int c;
```

```
a = 20;
```

```
b = 30;
```

```
pt1 = &a;
```

```
pt2 = &b;
```

```
c = (*pt1) + (*pt2)
```

We can perform addition & Subtractions on pointers but the both pointers should be same type

Accessing 1D array using pointer

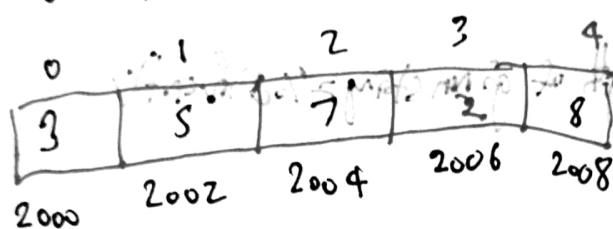
```
int *ptr;
```

```
ptr = &A[0];
```

```
printf("%d", ptr);
```

```
printf("%d", *ptr);
```

```
printf("%d", A[0]);
```



Q) WAP to read & display n integers array

```
void main() {
```

```
    int A[10];
```

```
    int n, i, sum = 0;
```

```
    int *ptr;
```

```
    ptr = A;
```

```
    printf("Enter no of elements");
```

```
    scanf("%d", &n);
```

```
    printf("Enter array elements");
```

```
    for (i = 0; i < n; i++) {
```

```
        scanf("%d", (ptr + i));
```

```
    }
```

```
    printf("array elements are");
```

```
    for (i = 0; i < n; i++) {
```

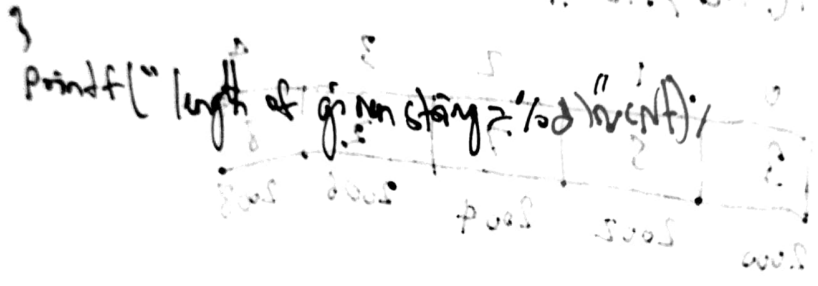
```
        printf("%d", *(ptr + i));
```


Q) WAP to find length of string using pointer

```

void main()
{
    char str[50] = "KITS WARANGAL";
    char *p1;
    int cnt = 0;
    p1 = str;
    while (*p1 != '\0')
    {
        cnt++;
        p1++;
    }
}

```



printf("length of given string = %d\n", cnt);

Output: length of given string = 13

Program End

Thank you

for your attention

Yours faithfully

[Signature]

(Name of the student)

(Roll No.)

(Date)

(Page No.)

(Teacher's Signature)

(The Principal's Signature)

Q) write to create an array and point sum of array elements using pointers

```
A) int a[20];
    int *ptr, n, i, s = 0;
    printf("enter n");
    scanf("%d", &n);
    printf("Enter array values");
    for(i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    for(i = 0; i < n; i++) {
        p = &a[0];
        sum = sum + (*p);
        p = p + 1;
    }
    printf("%d", s);
```

Benefits of pointers

- 1) More efficient for arrays & structures
- 2) Helps in passing of functions as arguments
- 3) Lets function change its calling arguments
- 4) Reduces length & time
- 5) Supports dynamic memory management

ptr++ → moves pointer forward

```

(a) void main() {
    char str[10] = "KTSW";
    char *p = str;
    while (*p != '\0') {
        printf("%c", *p);
        printf("%c", p);
        p++;
    }
}
    
```

Structures & pointers

It points to the address of a memory block where the structure is being stored

Declaration

struct structure *ptr;

Initialization of structure pointers

ptr = &structure-variable

Accessing Structure members using pointers

2 ways to access struct members

1) * or . Eg: (*s2).name

2) → Eg: s2 → name

No need of * in →

File Handling

File is a permanently stored form of data which is similar data

2 types of files

1) Text files → human-readable & plain English

2) Binary files → computer-readable & contains binary

Operations:

- Creating file
- Opening files
- Reading a file
- Writing to file
- Closing file

opening modes

r → read only

w → write only

a → append

rt → read & write

wt → read & write

at → read & write

File pointer

- We use file pointer to declare a file
- Pointer of file specifies next byte to be read or written to.

Declaration

FILE *fp;

- fopen() is used to open file

Eg :- fp = fopen("Hell.c", "r");

12/12/23

(111)

Writing to a file

We can add a single character into text file using `putc()`.

Eg:- `putc("x", fp1);`

There are 3 imp points to work with files in C:

- 1) File name
- 2) Data structure
- 3) Purpose

• First we have to declare a file pointer

```
FILE *fp1, *fp2;
```

• Then we have to open the file name in write mode

```
fp1 = fopen("first.c", "w");
```

• Use `putc` to write a char into the file

```
putc("x", fp1);
```

`putc` is a function to write a single character at a time to the file

getc

It is a method used to read a character from the file. `getc` can be used only on the file opened for reading purpose.

Eg: `ch = getch(fp2);`

EOF → End of file has to be closed using `fclose()`

Q) WAP to create a file

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *fp1;
```

```
    char ch1 = 'a';
```

```
    char ch2 = 'b';
```

```
    fp1 = fopen("ib2.c", "w");
```

```
    putc(ch1, fp1);
```

```
    putc(ch2, fp1);
```

```
    fclose(fp1);
```

```
}
```

Q) WAP to write to a file using user input

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *fp1;
```

```
    char ch1;
```

```
    fp1 = fopen("namdini.txt", "w");
```

```
    printf("Enter the data");
```

```
    while ((ch1 = getch()) != EOF) {
```

```
        putc(ch1, fp1);
```

```
    }
```

```
    fclose(fp1);
```

```
}
```

Q) WAP to read & display contents of file

```
int main() {
    FILE *fp1;
    char ch1;
    fp1 = fopen("text.txt", "r");
    printf("File data is \n");
    while ((ch1 = getc(fp1)) != EOF) {
        printf("%c", ch1);
    }
    fclose(fp1);
}
```

Q) WAP to copy contents of one file to another file

```
int main() {
    FILE *fp1, *fp2;
    char ch1;
    fp1 = fopen("file1.c", "r");
    fp2 = fopen("out.txt", "w");
    while ((ch1 = getc(fp1)) != EOF) {
        putc(ch1, fp2);
    }
    fclose(fp1);
    fclose(fp2);
}
```


getw

It is a method which can be used to read integers from a file

putw

It is a method which is used to write integers to a file